Introduction to the Theory of Computation

Reading

Sections 0.1, 0.2

Introduction

Let us start with a "simple" question?

> What is computation? What is involved in asking a computer to "compute" something?

Take a few moments to think about it.

You probably arrived at something roughly like this.

- Given a task at hand, a "problem", we write some sort of "program", or "algorithm", or series of instructions. We tell the computer about it.
- We now can "feed" an "input" into the program, and the computer will follow the instructions we gave it and end up with a result that tells us whether the input is appropriate for the problem.

Okay this sounds hopelessly vague. Let's try some concrete examples. Try to think of what steps you will need to take to solve the following problems. "Numbers" are here interpreted as a string of digits, so do not assume that you know about fancy things like arithmetic operations. Try to solve these problems with as little "memory" as possible: Your process should be reading one digit at a time, starting from the highest order digit, and it should try to maintain as little information as possible from the one digit to the next.

1. Given a number, determine if it is an even number
2. Given a number, determine if it is divisible by 3
3. Given a number, determine if it has more than four 5s in it
4. Given a number, determine if it is a palindrome (e.g. 125151521)
5. Given a string of parentheses, determine if the parentheses are "balanced", i.e. every open parenthesis closes.
6. Given a number, determine if it is "periodic", i.e. consists of a set of digits that then repeat in the same order a fixed number of times (more than once)

What you will notice after thinking about these problems, is that some require considerably more resources than others. Some don't require you to remember almost anything from one step to the next, others require you to remember very little, others still require you to remember a whole lot, possibly exhausting your available memory.

We will examine various models of computation that have progressively more power:

- **Finite Automata** consist of a finite number of "states", and all you can do on the next input is move from one state to another. With limited power but very simple to work with.
- **Pushdown Automata** also allow you to maintain a "stack" of stored values, from which you can add or remove from the top. But still contain a limited number of states. Powerful enough to parse most programming language programs.
- **Turing Machines** extend this further to provide an infinite number of states, as well as an infinite amount of space on which to write information. It is one of the richest models of computation known at this point.

In this course we will examine each of these models in detail, and look at their capabilities and limitations.